Daemon programming in Raspberry Pi

Jaakko Koivuniemi
CERN/PH

April 11, 2014

Abstract

command line interface, what is a daemon?, typical daemon structure

Command line interface and C

- used in computers long before graphical user interface
- usually enough for servers that do not need friendly user interface
- can be efficient over slow network connection
- in linux most important operations can always be done from CLI
- allows shell scripting and programming
- rapid developement of working programs

C get options

```c
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>

void printusage()
{
  printf("usage: numbers [-n N] [-h]\n");
}

int main(int argc, char **argv)
{
  int count=0;

  int optch=0;
  while(optch!=-1)
  {
    optch=getopt(argc,argv,"n:h");
    if(optch=='h')
```

```
19      {
          printusage();
21        return 0;
        }
23      if(optch=='n')
        {
25        count=atoi(optarg);
        }
27
    }
29
    int i=0;
31  for(i=0;i<count;i++)
    {
33    printf("%d\n",i);
    }
35
    return 0;
37  }
```

Compile and test

```
pi@raspberrypi ~ $ gcc -g -O -Wall numbers.c -o numbers
pi@raspberrypi ~ $ numbers -n 3
0
1
2
```

What is a daemon?

- general service run in background
- examples: atd, crond, httpd, ifplugd, ntpd, sshd, udevd

```
pi@raspberrypi ~ $ service --status-all
pi@raspberrypi ~ $ service ifplugd help
pi@raspberrypi ~ $ service ifplugd status
pi@raspberrypi ~ $ sudo service ifplugd stop
```

Writing a daemon program

- can read a configuration file from /etc/mydaemon_conf
- writes a log file to /var/log/mydaemon.log, verbose level can be set from configuration file
- logrotate(8) can be used to rotate/compress log file
- typically /var/lib/mydaemon/ has the live data
- /var/run/mydaemon.pid process number
- daemon needs to catch signals like SIGHUP, SIGTERM

- can be started and stopped manually with **service mydaemon start/stop**

- can be started or stopped by the system at different run levels

- can be written in C/C++ or as Perl/Python script, the first typically for mature projects and latter for fast developement with modern data structures

Example: temperature logging

- temperature from tmp102 read with I2C bus
- data written to a log file /var/log/tmp102d.log

```
void logmessage(const char logfile[200], const char message[200], int
  loglev, int msglev) {}

int read_data(int address, int length) { return 0}

void read_temp() {}

int cont=1; /* main loop flag */

void stop(int sig)
{
```

```
    sprintf(message,"signal %d catched, stop",sig);
12  logmessage(logfile,message,loglev,4);
    cont=0;
14}

16int main()
{
18  sprintf(message,"tmp102d v. %d started",version);
    logmessage(logfile,message,loglev,4);
20
    signal(SIGTERM,&stop);
22
    while(cont==1)
24  {
      read_temp();
26    sleep(60);
    }
28
    return 0;
30}
```

Daemonization

- fork off the parent process and let it terminate
- setsid - create a new session
- catch signals
- chdir - change the working directory of the daemon.
- umask - change the file mode mask according to the needs of the daemon
- close - close all open file descriptors that may be inherited from the parent process

Automatic start at boot time

- shell script at /etc/init.d/tmp102d

```
1 pi@raspberrypi ~ $  sudo update-rc.d tmp102d defaults
  pi@raspberrypi ~ $  sudo service tmp102d start/stop
```

References

- http://en.wikipedia.org/wiki/Daemon_(computing)
- http://stackoverflow.com/questions/17954432/creating-a-daemon-in-linux
- https://github.com/oh7bf/tmp102 (to be done)